Amendments to the Specification:

Please replace paragraph [0001] with the following amended paragraph:

[0001] This application claims priority to U.S. Provisional Application Serial No. 60/400,391 titled "JSM Protection," filed July 31, 2002, incorporated herein by reference. This application also claims priority to EPO Application No. 03291917.7, filed July 30, 2003 and entitled "Processor That Accommodates Multiple Instruction Sets And Multiple Decode Modes," incorporated herein by reference. This application also may contain subject matter that may relate to the following commonly assigned co-pending applications incorporated herein by reference: "System And Method To Automatically Stack And Unstack Java Local Variables," Serial No. [[ ]]10/632,228, filed July 31, 2003~~, Attorney Docket No. TI-35422 (1962-05401)~~; "Memory Management Of Local Variables," Serial No. [[ ]]10/632,067, filed July 31, 2003~~, Attorney Docket No. TI-35423 (1962-05402)~~; "Memory Management Of Local Variables Upon A Change Of Context," Serial No. [[ ]]10/632,076, filed July 31, 2003~~, Attorney Docket No. TI-35424 (1962-05403)~~; "A Processor With A Split Stack," Serial No. [[ ]]10/632,079, filed July 31, 2003~~, Attorney Docket No. TI-35425(1962-05404)~~; "Using IMPDEP2 For System Commands Related To Java Accelerator Hardware," Serial No. [[ ]]10/632,069, filed July 31, 2003~~, Attorney Docket No. TI-35426 (1962-05405)~~; "Test With Immediate And Skip Processor Instruction," Serial No. [[ ]]10/632,214, filed July 31, 2003~~, Attorney Docket No. TI-35427 (1962-05406)~~; "Test And Skip Processor Instruction Having At Least One Register Operand," Serial No. [[ ]]10/632,084, filed July 31, 2003~~, Attorney Docket No. TI-35248 (1962-05407)~~; "Synchronizing Stack Storage," Serial No. [[ ]]10/631,422, filed July 31, 2003~~, Attorney Docket No. TI-35429 (1962-05408)~~; "Methods And Apparatuses For Managing Memory," Serial No. [[ ]]10/631,252, filed July 31, 2003~~, Attorney Docket No. TI-35430 (1962-05409)~~; "Write Back Policy For Memory," Serial No. [[ ]]10/631,185, filed July 31, 2003~~, Attorney Docket No. TI-35431 (1962-05410)~~; "Methods And Apparatuses For Managing Memory," Serial No. [[ ]]10/631,205, filed July 31, 2003~~, Attorney Docket No. TI-35432 (1962-05411)~~; "Mixed Stack-Based

RISC Processor," Serial No. [[~~_____~~]]10/631,308, filed July 31, 2003~~, Attorney Docket No. TI-35433 (1962-05412)~~; "System To Dispatch Several Instructions On Available Hardware Resources," Serial No. [[~~_____~~]]10/631,585, filed July 31, 2003~~, Attorney Docket No. TI-35444 (1962-05414)~~; "Micro-Sequence Execution In A Processor," Serial No. [[~~_____~~]]10/632,216, filed July 31, 2003~~, Attorney Docket No. TI-35445 (1962-05415)~~; "Program Counter Adjustment Based On The Detection Of An Instruction Prefix," Serial No. [[~~_____~~]]10/632,222, filed July 31, 2003~~, Attorney Docket No. TI-35452 (1962-05416)~~; "Reformat Logic To Translate Between A Virtual Address And A Compressed Physical Address," Serial No. [[~~_____~~]]10/632,215, filed July 31, 2003~~, Attorney Docket No. TI-35460 (1962-05417)~~; "Synchronization Of Processor States," Serial No. [[~~_____~~]]10/632,024, filed July 31, 2003~~, Attorney Docket No. TI-35461 (1962-05418)~~; "Conditional Garbage Based On Monitoring To Improve Real Time Performance," Serial No. [[~~_____~~]]10/631,195, filed July 31, 2003~~, Attorney Docket No. TI-35485 (1962-05419)~~; "Inter-Processor Control," Serial No. [[~~_____~~]]10/631,120, filed July 31, 2003~~, Attorney Docket No. TI-35486 (1962-05420)~~; "Cache Coherency In A Multi-Processor System," Serial No. [[~~_____~~]]10/632,229, filed July 31, 2003~~, Attorney Docket No. TI-35637 (1962-05421)~~; "Concurrent Task Execution In A Multi-Processor, Single Operating System Environment," Serial No. [[~~_____~~]]10/632,077, filed July 31, 2003~~, Attorney Docket No. TI-35638 (1962-05422)~~; and "A Multi-Processor Computing System Having A Java Stack Machine And A RISC-Based Processor," Serial No. [[~~_____~~]]10/631,939, filed July 31, 2003~~, Attorney Docket No. TI-35710 (1962-05423)~~.

Please replace paragraph [0024] with the following amended paragraph:

[0024] The ALU 148 adds, subtracts, and shifts data. The multiplier 150 may be used to multiply two values together in one or more cycles. The instruction fetch logic 154 generally fetches instructions from instruction storage 130. The instructions first may be pre-decoded by the pre-decode logic ~~by~~ 158 and then decoded by decode logic 152. Because the JSM 102 may be adapted to process instructions from at least two

instruction sets, the decode logic 152 generally comprises at least two modes of operation, one mode for each instruction set. In particular, the decode logic unit 152 may include a "Java" mode in which Java instructions may be decoded and a "C-ISA" mode in which C-ISA instructions may be decoded. Of course, the modes depend on the particular instruction sets implemented which may be different from the instrument sets described herein. While decoding instructions from one instruction set in the corresponding mode, the decoding mode may be changed "temporarily" to permit a single instruction from the other opposite instruction set to be executed in the mode that corresponds to the other instruction set. The opposite instruction set instruction may be preceded by a byte such as the Java "Impdep1" byte to cause the temporary mode switches. In addition, the decoding mode may be changed "permanently" to permit a plurality of instructions from the opposite instruction set to be executed in the mode set that corresponds to the opposite instruction set. Such an instruction may be preceded by a byte such as a prefix. The prefix may be one or more implementation defined Java bytes, e.g., the Java "Impdep1" byte. The pre-decode logic 158 determines if an instruction is preceded by such bytes and considers this byte as an instruction prefix, as will be discussed below.

Please replace paragraph [0025] with the following amended paragraph:

[0025] The data storage 122 generally comprises data cache ("D-cache") 124 and data random access memory ("D-RAMset") 126. Reference may be made to copending applications U.S. Serial Nos. 09/591,537 filed June 9, 2000 ~~(atty docket TI-29884)~~, 09/591,656 filed June 9, 2000 ~~(atty docket TI-29960)~~, and 09/932,794 filed August 17, 2001 ~~(atty docket TI-31351)~~, all of which are incorporated herein by reference. The stack (excluding the micro-stack 146), arrays and non-critical data may be stored in the D-cache 124, while Java local variables, critical data and non-Java variables (e.g., C, C++) may be stored in D-RAM 126. The instruction storage 130 may comprise instruction RAM ("I-RAM") 132 and instruction cache ("I-cache") 134. The I-RAMset 132 may be used for "complex" micro-sequenced Bytecodes or other "micro-sequences or critical sequences

of codes," as will be described below. The I-cache 134 may be used to store other types of Java Bytecode and mixed Java/C-ISA instructions.

Please replace paragraph [0029] with the following amended paragraph:

[0029] Referring still to Figure 4, Bytecode A belongs to a first instruction set, e.g., a Java instruction. In order to determine the mode in which the decode logic 152 may operate for a next instruction, the pre-decode logic 158 preferably determines if any of Bytecodes B through F comprises a predetermined instruction indicating at least one succeeding instruction following Bytecode A belongs to the second instruction set, e.g., an instruction from the C-ISA set. The predetermined instruction may be a prefix to the succeeding instruction. A prefix, as used herein, is a Bytecode that indicates the type of instruction that follows the prefix. For example, a prefix (e.g., Impdep1) may be used as a prefix and may indicate that the instruction following the Impdep1 instruction belongs to a different instruction set. As such, by first determining if an instruction belongs to a particular instruction set, the decode logic 152 may <u>be</u> caused to adjust the decoding mode for decoding the instruction. Therefore, if the pre-decode logic 158 determines that Bytecode B is a Java Impdep1 instruction and the length of the currently decoded instruction is one byte corresponding <u>to</u> the Bytecode A, the decode logic 152 may be caused to skip the decoding of Bytecode B and switch from the first mode to the second mode to decode Bytecode C, where Bytecode C may be all or a portion of an instruction. After the decoding of the succeeding instruction that may include Bytecode C, the decode logic 152 switches back to the first mode. As such, the Java Impdep1 instruction may temporarily switch the decode logic from the first mode to the second mode to decode one instruction from the second instruction set and may switch back to the first mode after the decoding process of that instruction.

Please replace paragraph [0033] with the following amended paragraph:

[0033] The second instruction set may also include an instruction that permanently switches ~~from~~ the decode logic from the second mode to the first mode. In particular, the second instruction set may include a "SetJ" instruction indicating that a set of instructions,

e.g., a set of Java instructions, may follow. Upon detection of the SetJ instruction, the decode logic 152 permanently switches from the second mode (which may be used for decoding C-ISA instructions) to the first mode (which may be used for decoding Java bytecodes). The decode logic remains in the first mode until a Java Impdep1 instruction is detected.

Please replace paragraph [0011] with the following amended paragraph:

[0011] Figure 4 illustrates decoding an instruction in a first mode in parallel with pre-decoding a plurality of instructions; ~~and~~

Please replace paragraph [0012] with the following amended paragraph:

[0012] Figures 5 illustrates decoding in the second mode[[.]];

Please add the following new paragraph [0012.1]:

[0012.1] Figure 6A illustrates a method in accordance with at least some embodiments; and

Please add the following new paragraph [0012.2]:

[0012.2] Figure 6B illustrates a method in accordance with alternative embodiments.

Please add the following new paragraph [0036.1]:

[0036.1] Figure 6A illustrates a method in accordance with at least some embodiments. In this exemplary method, instructions are decoded in a given mode, wherein the mode may switch to another mode temporarily or permanently to allow other instructions to be executed. Specifically, the method starts (block 610) and moves to defining a first and second instruction set, wherein both instruction sets may comprise a Java bytecode (e.g., a Java Impdep1 bytecode) (block 612) that serves to define various instructions (e.g., a temporary instruction, a first permanent instruction, or a second permanent instruction). Thereafter, instructions from the first instruction set are decoded in a first mode, or instructions from the second instruction set are decoded in a second mode (block 614).

Whether decoding the first instruction set in a first mode or the second instruction set in a second mode, the method continues to detecting an instruction (e.g., the temporary instruction) indicating that one instruction or a plurality of instructions following the detected bytecode belong to the opposite instruction set (block 616). A determination is then made as to whether the one instruction or the plurality of instructions following the detected bytecode belong to the first or second instruction set (block 618). If the one instruction or the plurality of instructions belongs to the first instruction set, then the decode mode switches from the second mode to the first mode (block 620). On the other hand, if the one instruction or the plurality of instructions belongs to the second instruction set, then the decode mode switches from the first mode to the second mode (block 622). The switch between decode modes may be temporary switch allowing for the execution of one instruction from the opposite instruction set, or the switch may be a permanent switch allowing for the execution of a plurality of instructions from the opposite instruction set. Thereafter, the method ends (block 624).

Please add the following new paragraph [0036.2]:

[0036.2] Figure 6B illustrates a method in accordance with alternative embodiments. The method starts (block 630) and moves to defining a first instruction set comprising a temporary instruction and a second instruction set comprising both a temporary instruction and a first and second permanent instruction, wherein both instruction sets are executable by a processor of a mobile device (block 632). The mobile device may comprise a cellular telephone, for example. Thereafter, instructions from the first instruction set are decoded in a first mode or instructions from the second instruction set are decoded in a second mode, while concurrently pre-decoding appropriate instructions in the first or second instruction set (block 634). If decoding and pre-decoding of the second instruction set occurs, then in some embodiments the method proceeds to detecting the second permanent instruction (block 644). If the second permanent instruction is detected, then the decode mode permanently switches from the second mode to the first mode (block 650), and the method ends (block 652). In alternative embodiments, if decoding and pre-decoding of the second instruction set occurs, or if decoding and pre-decoding of the first

instruction set occurs, then the method proceeds to detecting a temporary instruction indicating that a subsequent instruction (or a subsequent plurality of instructions) belongs to the opposite instruction set (block 636). A determination is then made as to whether the subsequent instruction (or the subsequent plurality of instructions) belongs to the first or second instruction set (block 638). If the subsequent instruction (or the subsequent plurality of instructions) belongs to the first instruction set, then the decode mode temporarily switches from the second mode to the first mode (block 640), and the method ends (block 652). Alternatively, if the subsequent instruction (or the subsequent plurality of instructions) belongs to the second instruction set, then a determination is made as to whether the subsequent instruction is the first permanent instruction (block 642). If the subsequent instruction is the first permanent instruction, then the decode mode permanently switches from the first mode to the second mode (block 648), and the method ends (block 652). In other embodiments, if the subsequent instruction is not the first permanent instruction, then the decode mode temporarily switches from the first mode to the second mode (block 646), and the method ends (block 652).